RFC: New HDF5 API Routines for HPC Applications

Read/Write Multiple Datasets in a HDF5 file

Peter Cao

Quincey Koziol

Jonathan Kim

The HDF5 library allows a data access operation to access one dataset at a time, whether access is collective or independent. Accessing multiple datasets will require the user issuing an I/O call for each dataset. This RFC proposes new routines to allow users to access data in multiple datasets with single I/O call.

This RFC describes the new API routines, *H5Dread\_multi()* and *H5Dwrite\_multi()*, which perform a single access operation to multiple datasets in the file. The new routines can improve performance, especially in cases when data accessed across several datasets from all processes can be aggregated in the HDF5 or MPI-I/O library.

# Introduction

Parallel HDF5 (PHDF5) supports both independent and collective dataset access. When collective I/O is used, all processes that have opened the dataset may do collective data access by calling *H5Dread()* or *H5Dwrite()* on the dataset with the transfer property set for collective access. Accessing dataset collectively by using the MPIO VFD can improve I/O performance, [1] since data can be aggregated by MPI into large contiguous accesses to disk instead of small non-contiguous ones.

However the current HDF5 library does not support a single I/O call for accessing multiple datasets. For example, if you access five datasets in a file, you will need at least five I/O calls.

We propose to add two new functions to the HDF5 library: *H5Dread\_multi()* and *H5Dwrite\_multi()* . Using the proposed new read/write functions, users will be able to make a single function call to read or write data to multiple datasets in a HDF5 file. Note that the multiple datasets are located in the same HDF5 file for the scope of this task. The new functions can be used for both independent and collective I/O access, but the main purpose of this task is for utilizing the collective I/O case.

# Use Case

## Improving FLASH I/O for an ANL project

FLASH code was designed to simulate thermonuclear flashes on a Cartesian, structured mesh. The mesh consists of cells that contain physical quantities such as density, pressure and temperature (also known as mesh variables). Each cell is assigned to a self-contained block. In the FLASH file layout, a block is stored in an HDF5 file and mesh variables are stored as 4D datasets in the file.

In a FLASH simulation, the time spent on file I/O is a common bottleneck. Using collective I/O[1] improves I/O performance for HPC applications like FLASH. Current parallel HDF5 performs collective I/O on a single dataset and requires many I/O calls in FLASH simulations since there are frequently many variables accessed during each time step. Using the proposed collective I/O on multiple datasets will reduce the number of I/O calls. In an experimental study, Rob Latham, Chris Daley, etc.[2] have showed that the average time for writing a file is reduced by half when collective I/O on multiple variables is used:

“*The standard file layout approach (storing application data in multiple library objects), however, offers a slight performance trade-off. Each function call represents a relatively expensive I/O operation. All other factors aside, if the goal is to achieve the highest I/O performance a better approach would describe the entire application I/O pattern and then execute a single call. If the application places all mesh variables into a single I/O library object, as in the experimental file layout approach, then a single I/O library call could be issued to service all application variables instead of N separate calls. Experiments confirm that this approach does improve performance*.”[2]

<TBD> CGNS User case: Quincey help

<TBD> More specific explains for Requirement management section

* For the purpose of test cases (Albert said he has an example)
* Specify if user can help to run the test and send us those before & after results to verify the performance improvements.

# Implementation

The basic approach for multi-dataset collective I/O is similar to the POSIX lio\_listio() call, which takes a list of buffers, offsets and lengths to perform series of read and write operations on a file in a single call. The major difference from the typical HDF5 API call is that the new routines add information from multiple datasets to the I/O mapping list and construct larger MPI derived datatypes for collective I/O operations for read and write operations in separate manner. Internally, the multi-dataset implementation will be similar to the current implementation of collective chunk I/O on a single dataset.

The following example chart shows the conceptual implementation approach for the new API functions.



Note that sorting the list by file addresses is necessary because MPI requires the file type to consist of derived data types whose displacements are monotonically non-decreasing.

## New API Functions

Two new functions, *H5Dread\_multi()* and *H5Dwrite\_multi()* are proposed here.

A common type structure is defined as below and used for keeping multiple dataset’s information for both APIs.

typedef struct H5D\_rw\_multi\_t

{

hid\_t dataset\_id;

hid\_t file\_space\_id;

void \* rbuf; /\* data buffer for read \*/

const void \* wbuf; /\* data buffer for write \*/

hid\_t mem\_type\_id;

hid\_t mem\_space\_id;

} H5D\_rw\_multi\_t;

### H5Dread\_multi()

The API function description is as shown below.

herr\_t H5Dread\_multi(hid\_t file\_id,

size\_t count,

H5D\_rw\_multi\_t info[],

hid\_t dxpl\_id);

Parameters:

* file\_id: file or group id for the location of datasets
* count: the number of accessing datasets.
* Info: the array of dataset information and read buffer.
* dxpl\_id: dataset transfer property.

Return:

* a non-negative value if successful; otherwise returns a negative value.

This routine performs collective or independent I/O reads from multiple datasets. In collective mode, all process members of the communicator associated with the HDF5 file must participate in the call.

Each process creates the information required to perform each read in the array of H5D\_rw\_multi\_t structures, and passes the array through to *H5Dread\_multi()*.

Brief description for internals after being called:

* Each process obtains the list of dataset information from the info[] array structure, and constructs a MPI derived datatype describing the sections from multiple datasets in a HDF5 file to be read.
* All processes end up calling *MPI\_File\_read\_at\_all()* once each for collective I/O or *MPI\_File\_read\_at()* once each for independent I/O.
* Each process tidies up, and then returns with the desired data into the buffer of the info[] array structure.

When an application issues the multi-read call, *H5D\_rw\_multi\_t* array elements are expected to be different among processes that are participating in the collective operation due to different selections. This means that not only the actual data in the buffers can be distinct (like most collective I/O operations), but the dataset (dataspaces, datatypes, etc…) values for every process can be distinct.

All processes are required to pass the same *file\_id* and the same property values for the *dxpl\_id*.

Refer to the example section for better understanding of usage.

The same rule applies to *H5Dwrite\_multi()* that is detailed in the following section.

### H5Dwrite\_multi()

The API function description is as shown below.

herr\_t H5Dwrite\_multi(hid\_t file\_id,

size\_t count,

H5D\_rw\_multi\_t info[],

hid\_t dxpl\_id);

Parameters:

* file\_id: file or group id for the location of datasets
* count: the number of accessing datasets.
* Info: the array of dataset information and write buffer.
* dxpl\_id: dataset transfer property.

Returns:

* a non-negative value if successful; otherwise returns a negative value.

This routine performs collective or independent I/O writes to multiple datasets. In collective mode, all process members of the communicator associated with the HDF5 file must participate in the call.

Each process creates the information required to perform each write in the array of H5D\_rw\_multi\_t structures, and passes the array through to *H5Dwrite\_multi()*.

Note that when overlapping selections are used, the data stored in the file for the overlapping regions is undefined (as is the case for H5Dwrite).

Brief description for internals after being called:

* Each process obtains the list of dataset information from the info[] array structure, and constructs a MPI derived type describing the sections from multiple datasets in a HDF5 file to be written.
* All processes ends up calling *MPI\_File\_write\_at\_all()* once each for collective I/O or *MPI\_File\_write\_at()* once each for independent I/O.

## Example cases

This examples are based on assumption that using multi read API on a HDF5 file with four datasets, ‘d1’, ‘d2’, ‘d3’ and ‘d4’. Using multi write API would be practically same.

Pseudo code is used to show how the API can be used in simplified manner focusing on this task’s scope.

### Example1: all processes read from same datasets ‘d1’, ’d2’ and ‘d3’

Consider the following as an example running with 2 processes:

* Rank 0 process (P0) reads data portions from datasets ‘d1’, ‘d2’, and ‘d3’.
* Rank 1 process (P1) reads data portions from datasets ‘d1’, ‘d2’ and ‘d3’.

Chart view:

A HDF5 file

**P0**

**P1**

**P1**

d2

d3

d4

d1

**P1**

**P0**

**p0**

Pseudo code below:

|  |
| --- |
| Open datasets ‘d1’, ‘d2’ and ’d3’  Make selections from each dataset.  Set ‘dxpl’ for collective operation.  size\_t count = 3 /\* three datasets \*/  If (mpi\_rank == 0) /\* P0 \*/  H5D\_rw\_multi\_t info[3] = { {d1’s P0 select…}, {d2’s P0 select…}, {d3’s P0 select…} }  If (mpi\_rank == 1) /\* P1 \*/  H5D\_rw\_multi\_t info[3] = { {d1’s P1 select…}, {d2’s P1 select…}, {d3’s P1 select…} }    H5Dread\_multi (file\_id, count, info, dxpl) /\* get data via info’s buf \*/ |

### Example2: each process read from different datasets or none

Consider the following as an example running with 3 processes:

* Rank 0 process (P0) reads data portions from datasets ‘d1’, ‘d2’, and ‘d3’.
* Rank 1 process (P1) reads data portions from datasets ‘d3’ and ‘d4’.
* Rank 2 process (P2) does not read anything.

Chart view:

A HDF5 file

d4

d3

d2

d1

**P1**

**P1**

**P0**

**P0**

**P0**

Pseudo code below:

|  |
| --- |
| Open datasets ‘d1’, ‘d2’,’d3’ and ‘d4’  Make selections from each dataset.  Set ‘dxpl’ for collective operation.  If (mpi\_rank == 0) /\* P0 \*/  count = 3; /\* three datasets \*/  info[3] = { {d1’s P0 select…}, {d2’s P0 select…}, {d3’s P0 select…} }    If (mpi\_rank == 1) /\* P1 \*/  count = 2; /\* two datasets \*/  info[2] = { {d3’s P1 select…}, {d4’s P1 select…} }  If (mpi\_rank >= 2) /\* P2 \*/  count = 0 /\* no dataset access \*/  Info = NULL  H5Dread\_multi (file\_id, count, info, dxpl) /\* get data via info’s buf \*/ |

# Future Consideration

According to some discussions, we may be able to consider developing H5Dcreate\_multi(), H5Dopen\_multi() and H5Dclose\_multi() APIs in the future as separate tasks if necessary or requested by user.

[1] Yang M and Koziol Q, 2006. Using collective IO inside a high performance IO software package—HDF5 Technical Report National Center of Supercomputing Applications

[2] Rob Latham, Chris Daley, etc., March 2012. A case study for scientific I/O: improving the FLASH astrophysics code, <http://iopscience.iop.org/1749-4699/5/1/015001/article>

# Revision History

|  |  |
| --- | --- |
| *August 28, 2012:* | Version 1 by Peter Cao. Circulated internally. |
| *Sep 27, 2012:* | Version 2: updated based on internal reviews. |
| Feb 15, 2013: | Version 3: Updated based on internal reviews. Revised APIs and related contents.  The task entry is HDFFV-8313 in JIRA. |
| March 04, 2013: | Version 3.1: Updates based on internal reviews. More updates and add example section. |
| March 07, 2013: | Version 3.2: Some minor updates. Add chart view in example section. |
| March 12,2013 | Version 3.3: Some updates from internal presentation on 03-08-2013. |
|  |  |
|  |  |